METHOD AND APPARATUS FOR MANAGING CONCURRENT ACCESS TO MULTIPLE MEMORIES

Inventors: Lee E. Johnson, Jr., Round Rock; Daryl J. Kokoszka; Steven P. Larky, both of Austin, all of Tex.

Assignee: International Business Machines Corporation, Armonk, N.Y.

Filed: Sep. 27, 1994

Related U.S. Application Data


Field of Search

395/122, 395/163

References Cited

U.S. PATENT DOCUMENTS

3,742,457 6/1973 Calle et al. 395/275
4,033,434 12/1976 Scheuneman 395/550
4,037,210 7/1977 Sharp 395/275
4,038,642 7/1977 Bouknecht et al. 395/275
4,133,089 1/1980 Daughton et al. 395/275
4,309,755 1/1982 Lanty 395/275
4,318,172 3/1982 Yamada et al. 395/250
4,400,788 12/1984 Rasmussen 395/275
4,521,850 6/1985 Wilhite et al. 395/275
4,538,144 8/1985 Yamagami 340/747
4,549,263 10/1985 Calder 395/275
4,550,386 10/1985 Hiroseawa et al. 395/275
4,567,038 4/1986 Sims et al. 340/729
4,600,986 7/1986 Scheuneman et al. 395/425
4,609,917 9/1986 Shen 340/729

ABSTRACT

A memory apparatus includes a circuit for receiving and serially storing a plurality of instructions and a plurality of buffer memories each including a buffer controller for regulating access to that buffer. Also included is a circuit, connected to each buffer controller and the receiving circuit, for accessing one or more of said buffers in response to a first serially stored instruction while, in response to at least one other serially stored instruction, concurrently accessing at least one remaining buffer.

18 Claims, 16 Drawing Sheets
<table>
<thead>
<tr>
<th>Patent Number</th>
<th>Date</th>
<th>Inventor(s)</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,511,154</td>
<td>3/1987</td>
<td>Dines et al.</td>
<td>364/521</td>
</tr>
<tr>
<td>4,648,049</td>
<td>6/1987</td>
<td>Brown et al.</td>
<td>364/521</td>
</tr>
<tr>
<td>4,682,297</td>
<td>7/1987</td>
<td>Iwami</td>
<td>364/521</td>
</tr>
<tr>
<td>4,691,295</td>
<td>9/1987</td>
<td>Erwin et al.</td>
<td>395/425</td>
</tr>
<tr>
<td>4,697,178</td>
<td>1/1987</td>
<td>Heckel</td>
<td>340/729</td>
</tr>
<tr>
<td>4,725,831</td>
<td>2/1988</td>
<td>Coleman</td>
<td>340/747</td>
</tr>
<tr>
<td>4,750,113</td>
<td>6/1988</td>
<td>Buggert</td>
<td>395/275</td>
</tr>
</tbody>
</table>
FIG. 3

\[
\begin{array}{c|c}
R_0 + \triangle R_s & R_0 + \triangle R_s + \triangle R_x \\
G_0 + \triangle G_s & \vdots \\
B_0 + \triangle B_s & \vdots \\
Z_0 + \triangle Z_s & X_L + \triangle X_L + 1 \\
X_L + \triangle X_L & X_L + 2 \triangle X_L + 1 \\
\end{array}
\]

FIG. 11
dy = (y2 - y1), dx = (x2 - x1)

plt process

bpg_00

dy > 1 & dx > 1

dy > dx = slope_gt_one

slopec_gt one

x1/5 φ 2

if slope_gt_one then
dterm = 2dx - dy else
dterm = 2dy - dx

x2/5 φ 1

i2 = 2(dx - dy), i1 = 2dx

else
i2 = 2(dy - dx), i1 = 2dy

x2/5 φ 2

stbybr

Handoff latched parameters and flags.

FIG. 7
FIG. 9

from Rasterizer front end processor

Line/BLIT Busy
Misc. Busy
Triangle Busy
Data Path Busy

Send Triangle Write Strobe if Triangle request and not Triangle Busy.
Send Line/BLIT Write Strobe if Write/BLIT request and Not Write/BLIT Busy, ETC.

Write Strobes to rasterizer front end processor

 INTERFACE (Iordy) Ready
Input R/W
Input Strobe (Iostrb) to/from Graphics front end Processor

 TRIANGLE REGISTER REQUESTS
LINE/BLIT REGISTER REQUESTS
DATA PATH REGISTER REQUESTS
MISC. REGISTER REQUESTS

SELECT ONE OF 2n REGISTER REQUESTS
FIG. 10

**REGISTER CATEGORIES:**
- Overall request
- Triangle request
- Line/BIT request
- Data path request

**REGISTER IS NOT READY IF:**
- Overall request and overall not ready
- Triangle request and triangle not ready
- Line/BIT request and Line/BLIT not ready
- Data path request and Data path not ready

**READ CYCLE**
Enable OCD's output data from Register being read

**WRITE CYCLE**
Send write strobe to register

send Iordy

state 1

send Iordy

state 0

Iosrb received
hold Address
hold R/W

Iosrb not received
decode address
decide which Register category

state 3

send Iordy if register ready

register ready

state 2

FIG. 10
FIG. 13
Functional Block

300 ～ Input Interface

340 ～ Command Inter

310 ～ Triangle Calc

320 ～ Parameter Calc

360 ～ Draw Sequencer

380 ～ Memory Interf

330 ～ Format Convert

FIG. 17
METHOD AND APPARATUS FOR MANAGING CONCURRENT ACCESS TO MULTIPLE MEMORIES

This is a continuation of application Ser. No. 08/078,950 filed Jun. 16, 1993, which is a continuation of application Ser. No. 07/614,355, filed Nov. 15, 1990.

RELATED PATENT APPLICATIONS


TECHNICAL FIELD

The present invention relates generally to computer graphics cards and more specifically to a high performance rasterization processor.

BACKGROUND ART

FIG. 1 is a block diagram of a typical scalar computer. The computer includes a main processor coupled to a memory, an input device and an output device. Input device may include a keyboard, mouse, tablet or other type of input devices. Output device may include a text monitor, plotter or other type of output devices. The main processor may also be coupled to a graphics output device such as a graphics display through a graphics card. The graphics card receives instructions regarding graphics from the main processor.

DISCLOSURE OF THE INVENTION

In accordance to the present invention a memory apparatus is provided that includes a circuit for receiving and serially storing a plurality of instructions and a plurality of buffer memories each including a buffer controller for regulating access to that buffer. Also included is a circuit, connected to each buffer controller and the receiving circuit for accessing one or more of the buffers in response to a first serially stored instruction while, in response to at least one other serially stored instruction, accessing at least one remaining buffer.

A further understanding of the nature and advantages of the present invention may be realized by reference to the remaining portions of the specification and the drawings.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a block diagram of a typical scalar computer;
FIG. 2 is a block diagram of a preferred rasterization processor and display controller;
FIG. 3A is a diagram which shows the relationship between the parameters in the triangle data structure and the rendered triangle primitive;
FIG. 4 is a diagram of a preferred embodiment of the triangle interpolation subsystem;
FIG. 5 is a timing diagram which shows the parallelism achieved by the triangle interpolator;
FIG. 6 is a block diagram of the bitblit and line draw parameter calculator dataflow;
FIG. 7 is a state flow diagram of the bitblit and line draw parameter calculator;
FIG. 8 is a block diagram of the bitblit and line draw sequencer dataflow;
FIG. 9 is a more detailed block diagram of the input interface shown in FIG. 2;
FIG. 10 is a state diagram of the input interface shown in FIG. 9;
FIG. 11 is a more detailed block diagram of the memory hypervisor shown in FIG. 2;
FIGS. 12 and 13 show how the memory hypervisor of FIG. 2 would process a series of writes to the frame buffer; and
FIGS. 14 through 17 are timing diagrams that show how the rasterization processor and display controller of FIG. 2 would process various types of image primitives.

BEST MODE FOR CARRYING OUT THE INVENTION

In a preferred embodiment of the invention, graphics card includes a front end graphics processor, a rasterization processor and display controller (also known as a rasterizer or a rasterization engine), a frame buffer, an attribute buffer and a RAMDAC. The frame buffer and the attribute buffer are usually VRAM and the Z buffer is usually DRAM. The front end graphics processor receives instructions from the main processor regarding a graphics construct or image in world coordinates and described generally by what are called image primitives which include triangles, bitblits, line etc. The front end graphics processor can also handle pixel instructions from the workstation processor. The front end graphics processor then performs various transformation, clipping and lighting instructions thereby describing the graphics construct or image in near screen coordinates. This information is then passed on to rasterizer.

Rasterizer performs an inner loop of the processing of a graphics image or construct. The rasterizer typically generates pixels along a line primitive, copies from one area on a display to another, performing a logical or arithmetic function between two areas on a display (such as an exclusive-OR bitblit), filling a triangle shaped area with shaded pixels after performing a depth comparison test (Z buffered Gouraud shaded triangles), and other typical graphics functions. Overall, the rasterizer updates the frame buffer, Z buffer and attribute buffer in screen coordinates based on the information from the front end graphics processor and the information previously stored in the buffers.

The frame buffer, which may be double buffered, includes a description of the red, green and blue colors for each pixel of the graphics display. The Z buffer contains a description of the depth or Z value of the pixels stored in the frame buffer. This information is useful for the rasterizer to determine whether or not a proposed pixel to be displayed based on a logical test. If the logical test is false, then the frame and Z buffers are not updated and the current pixel is displayed rather than the proposed pixel. If the logical test is true, then the frame and Z buffers are updated and the
proposed pixel is displayed. In the preferred embodiment, the frame buffer has 24 planes (8 planes each for red, green and blue) for 1280 by 1024 pixels and the Z buffer has 24 bits per pixel for the same number of pixels. The attribute buffer 250, which may also be double buffered, includes control information about the pixels in the frame buffer such as which windows, overlays and microcode plane masks they belong to, whether they are write protected, etc.

RAMDAC 260 then uses the description of the pixels in the frame buffer and the control planes from the attribute buffer to generate an analog RGB signal stream. Graphics display 150 then displays the graphical image using the RGB signal stream.

FIG. 2 is a block diagram of rasterization processor 220. An input interface 300 communicates with the front end graphics processor in a predetermined protocol. Based on this communication, the input interface decodes addresses, directs data to the appropriate internal registers, and generates interface timings which meet a rasterizer front end 305's requirements. For example, the input interface sends information pertaining to triangle primitives to a triangle interpolator 310. In addition, the input interface sends the address information of bitblt and line primitives to a bitblt and line draw parameter calculator 320 and the corresponding data to a width and format converter 330. In the preferred embodiment, the input interface determines if the internal registers are free to receive data. If the internal registers are not available, then the input interface write protects the internal registers and holds up the data until the appropriate internal registers are free to receive the data. In addition, the input interface read protects internal registers that do not have valid data. The input interface has two-way communication, including status information, with a command interpreter and supervisor 340. The input interface also passes certain CRT control codes to a CRT controller 350.

Triangle interpolator 310 can be viewed as an address and data splitter. Information pertaining to triangle primitives are sent to the triangle interpolator 310, which the triangle interpolator then separates into a series of line primitives. The triangle interpolator then splits the address information from the data for the line primitives and passes the address information to the bitblt and line draw parameter calculator. The triangle interpolator also passes the corresponding data (including color and x information) to a data path 370 in the appropriate width and format. The triangle interpolator can also handle trapezoid primitives (quadrilaterals that have upper and lower edges that are parallel to the X axis) in a similar manner.

Bitblt and line draw parameter calculator 320 receives the address information of horizontal line primitives from triangle interpolator 310 and the address information of bitblt and line primitives in any orientation from the graphics processor via input interface 300. The bitblt and line draw parameter calculator then calculates various parameters for the bitblt and line primitives and passes that information to a bitblt and line draw sequencer 360. The bitblt and line draw sequencer then generates incremental pixel addresses for every pixel affected by the bitblt or line primitive. In addition, the sequencer also provides controlling information to data path 370. This information is then passed on to a hierarchical memory controller 380 (also referred to as a memory hypervisor) to update the frame buffer and the Z buffer.

Width and format converter 330 translates input data from the front end graphics processor or main processor to a format compatible to the target buffer. The converter handles strings of data for pixels that are contiguous. The data includes color and Z information. Lines with interpolated color signals do not get passed to the width and format converter, but are passed directly to the bitblt and line draw parameter calculator 320. In the preferred embodiment, the converter can convert 1 bit pixels to 24 bit pixels allowing expansion of monochrome text to any of two colors in the frame buffer. In addition, the width and format converter 330 can convert 32 bit words to 40 bit words or other width and format conversions necessary for the utilization of the rasterizer. Once the width and format of the incoming data is converted, the data is passed to data path 370.

Data path 370 passes data from triangle interpolator 310 and the width and format converter 330 to the memory hypervisor 380 in cooperation with the bitblt and line draw sequencer 360. This cooperation is coordinated by the command interpreter and supervisor 340. The data path also performs all necessary data manipulation such as pixel rotation and alignment, as well as source and destination mixing. Command interpreter and supervisor 340 coordinates the action of rasterizer 220 through various control signals. The command interpreter and supervisor also communicates the status of the rasterizer to the graphics processor via the input interface 300.

CRT controller 350 handles various control information for the CRT or display device. This includes horizontal and vertical sync refresh, frame blanking, composite sync, and other control signals that coordinate the actions of the graphics display device through memory hypervisor 380. The CRT controller also generates the addresses and load signals for the operations of the frame buffer.

Memory hypervisor 380 handles updating the frame buffer, Z buffer, and attribute buffer via frame buffer controller 390, Z buffer controller 392, and attribute buffer controller 394, respectively. The attribute buffer control planes are updated by the memory hypervisor as instructed by the graphics processor. The frame buffer pixels are updated according to a Z algorithm. That is, a logical test is performed. If the logical test is true, then the frame buffer and the Z buffer are updated by the new pixel values by the memory hypervisor 380. This Z algorithm is different from a Painter's algorithm wherein the Z values of primitives, rather than pixels, are compared to determine which primitive is in front of the other for display purposes.

The advantage of the separation of bitblt and line draw parameter calculator 320 and the sequencer 360 is that once the parameters are calculated and passed on to the sequencer, the parameter calculator is then able to handle the next bitblt or line primitive without waiting for the line draw sequencer to complete its processes. This is particularly useful for handling triangle and trapezoid primitives. That is, the triangle interpolator 310 splits the triangle primitives into a series of horizontal line primitives which are passed sequentially to the bitblt and line draw parameter calculator. The triangle interpolator can then move on to the next triangle line while the bitblt and line draw parameter calculator and the bitblt and line draw sequencer go through their processes for handling the various horizontal lines.

In the preferred embodiment, the triangle interpolator works at a speed of ten cycles per line primitive split from a triangle or trapezoid primitive, the bitblt and line draw parameter calculator works at a speed of six cycles per line primitive, and the bitblt and line draw sequencer works at a speed of one cycle per pixel. As a result, the length of the line primitives (i.e., the number of pixels) drawn by the bitblt
and line draw sequencer directly affects how frequently the sequencer can start drawing a line primitive such that either the parameter calculator or the sequencer is the one holding up the other. When drawing triangle primitives, the width of the triangle primitive determines the length of the resulting line primitives. If you use smaller and smaller triangle primitives, then the bitblt and line draw sequencer can keep up with the parameter calculator. This is extremely useful for small triangle calculations such as Gouraud shading. If you tessellate down to one triangle primitive per pixel, then that is roughly equivalent to Phong shading.

The individual subsystems of the rasterizer will now be shown to more fully describe the present invention. Subsequently, various timing diagrams will be shown to more fully describe how the various subsystems interrelate.

Triangle Interpolator

The triangle interpolator can be viewed as an address and data splitter. For example, a Gouraud-shaded triangle can be rendered by slicing the triangle primitive into horizontal lines, interpolating along the left edge to calculate the XY values and colors for each line's start, and the right edge to find each line's end point. The triangle interpolator is given the following 18 parameters from the input interface:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ymin</td>
<td>y value of the top of the triangle</td>
</tr>
<tr>
<td>Ymax</td>
<td>y value of the bottom of the triangle</td>
</tr>
<tr>
<td>Xl</td>
<td>starting x value of the left side of the triangle</td>
</tr>
<tr>
<td>Xr</td>
<td>starting X value of the right side of the triangle</td>
</tr>
<tr>
<td>Sx</td>
<td>change in the X value of the left side of the triangle for every y</td>
</tr>
<tr>
<td>Sy</td>
<td>change in the y value of the right side of the triangle for every y</td>
</tr>
<tr>
<td>R0, G0, B0, Z0</td>
<td>initial color and depth values, defined at [Ymin, Xleft]</td>
</tr>
<tr>
<td>SR, SG, SB, SZ</td>
<td>color and depth deltas along the left slope and x axis, respectively</td>
</tr>
</tbody>
</table>

From these 18 input parameters, the triangle interpolator generates the following 11 output parameters for triangle to be rendered, 7 of which are passed for each horizontal line primitive to be drawn:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>vertical address for the horizontal line</td>
</tr>
<tr>
<td>Xl</td>
<td>starting x value of the left end of the line</td>
</tr>
<tr>
<td>Xr</td>
<td>starting X value of the right end of the line</td>
</tr>
<tr>
<td>R</td>
<td>initial red for line</td>
</tr>
<tr>
<td>G</td>
<td>initial green for line</td>
</tr>
<tr>
<td>B</td>
<td>initial blue for line</td>
</tr>
<tr>
<td>SR</td>
<td>delta red for line</td>
</tr>
<tr>
<td>SG</td>
<td>delta green for line</td>
</tr>
<tr>
<td>SB</td>
<td>delta blue for line</td>
</tr>
<tr>
<td>Z</td>
<td>initial x for line</td>
</tr>
<tr>
<td>EZ</td>
<td>delta x for line</td>
</tr>
</tbody>
</table>

FIG. 3B is a diagram of a preferred embodiment of the triangle interpolation subsystem that is used to calculate the output parameters from the input parameters. Triangle interpolator 310 includes a buffer (register file) management controller 400 (TFB_FSM), a triangle interpolation and parameter handoff controller 401 (TRI_FSM), an address mapper 402, a comparator 403, thirty six 32-bit words of register file storage 404, a carry look-ahead adder 405 for interpolation and its accumulator register 406, and data buses 407 to line generation subsystems (addressing, Color and Z interpolation). The address mapper is a lookup table which creates the physical register file addresses by decoding the inputs from buffer management controller 400, handoff controller 401 and input interface 300. The comparator 403 detects an end-of-triangle condition. Carry look-ahead adder 405 is used for interpolation.

Register file 404 has enough storage capacity to hold parameters for two triangle primitives. This allows triangle operations to be doubled buffered. That is, data for a new triangle is loaded into an idle portion of the buffer while the interpolation controller 401 utilizes the other half of the buffer. Buffer management controller 400 insures that the two buffers are kept separate. The register file has two write and three read data ports. One write port (W0) of the register file as assigned for loading new data. The other write port (W0) is used by the interpolation controller two store computation results. The new data can be loaded into the idle buffer and parallel with triangle interpolation tasks until the idle half of the idle buffer is full. The three read data ports are used for parameter passing and computation.

The TRI_FSM algorithm used in the interpolation controller 401 is tuned to further increases throughput. The initial cycles of line generation need only the address information created by the triangle pass. The color parameters are not needed until line drawing actually begins. To take advantage of this, the triangle computation operations are sequenced such that the line addressing parameters are generated first then the color and depth parameters next. At the beginning of the interpolation pass, the line generator is checked for availability. If the check is positive, a parameter...
passing sequence different from the one described above is used. The address parameters are passed off one at a time directly from the accumulator and line sequencing is started while generation of color and depth parameters continues. The color and depth parameters are then passed at a later time. The parameter calculator 320 then processes the line addresses while the triangle interpolator continues processing color and depth information. The triangle controller can immediately begin the next pass or even a new triangle if the current triangle is complete and the idle buffer has been filled.

FIG. 5 is a timing diagram that shows the parallelism achieved by the triangle interpolator. For purposes of clarity, a state diagram of TR2_FSM is included. The horizontal axis represents increasing time. The vertical axis indicates the tasks being performed in each cycle. For example, from time T0 to time T1 the three line addressing parameters (V0, XL0, XR0) are being transferred from the register file to the line draw parameter calculator while three new register file addresses are being clocked into latches in preparation for a jump to time T2. At time T4 to time T5 the three delta color values are being transferred while the read address needed for incrementing Y is being prepared. At time T8 to time T9, XR is being stored, Zi is being computed and the addresses for computing Ri are being prepared.

Bibbit and Line Draw Parameter Calculator

The bibbit and line draw parameter calculator 320 pre-processes addressing information for the bibbit and line draw sequencer 340. The sequencer 360 computes individual pixel addresses, controls color interpolation pacing and communicates with the memory hypervisor. By partitioning memory addressing into two tasks (an address preprocessing task and a pixel addressing task, each task to be performed by separate subsystems), a first line or bibbit need only be partially processed prior to starting processing on a second line or bibbit. This also applies to both Bresenham (see pages 74 to 81 of the second edition of FUNDAMENTALS OF INTERACTIVE COMPUTER GRAPHICS by J. D. Foley and A. Van Dam) and DDA (see pages 73 to 74 of the second edition of FUNDAMENTALS OF INTERACTIVE COMPUTER GRAPHICS by J. D. Foley and A. Van Dam) line calculations. This is particularly useful when rendering triangle primitives.

The 4 line input address parameters are processed to produce line direction and quadratic information, as well as the Bresenham algorithm parameters. The bibbit inputs must be processed to determine which direction in X and Y that pixel reading and writing must proceed such that pixels are not written over before being copied. The input and output parameters for line primitives are:

<table>
<thead>
<tr>
<th>Line Inputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y1: y value at first endpoint</td>
</tr>
<tr>
<td>Y2: y value at second endpoint</td>
</tr>
<tr>
<td>X1: x value at first endpoint</td>
</tr>
<tr>
<td>X2: x value at second endpoint</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Line Outputs:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ystart:</td>
</tr>
<tr>
<td>Yend:</td>
</tr>
<tr>
<td>Xstart_quot: X1 divided by five</td>
</tr>
<tr>
<td>Xend_quot: X2 divided by five</td>
</tr>
<tr>
<td>Xstart_rem: Remainder from division of X1/5</td>
</tr>
<tr>
<td>Xend_rem: Remainder from division of</td>
</tr>
</tbody>
</table>

The parameter calculator enables a mix of graphics primitives in any sequence. For example, a bibbit can precede a DDA line, which can then precede a sequence of polyline segments, a series of triangle spans, etc. An idle state of the parameter calculator allows initialization to occur, thereby not wasting any cycles. In addition, an initial state of the parameter calculator, where output parameters are passed to the sequencer, performs the final parameter calculations. As a result, the last output values are passed directly to the sequencer rather than to registers within the parameter calculator. This subsystem can also perform up to five arithmetic operations simultaneously during some processing steps [including (a+b), (a−b), (b−a), (a−b)+2, a−b−2, a+2, a−b, a+b, a/5] to maintain high throughput.

FIG. 6 is a block diagram of the bibbit and line draw parameter calculator dataflow. The input parameters are received from input interface 300 and triangle interpolator 310. The output parameters are then calculated from the input parameters by multiplexers 510, inverters 520, carry look-ahead adders 530, divider 540 and registers 550.

FIG. 7 is a state flow diagram for the parameter calculator 320, with detail and emphasis on a Bresenham line sequence. This controller switches the multiplexers and registers depicted in FIG. 6 to complete the parameter calculation sequence. For example, step slp_gt one flag is set. In step d_term, the slope_gt_one flag is tested to determine how the dterm
variable is calculated, as shown in the comments accompanying the figure.

Bibbit and Line Draw Sequencer.

The bitblt and line draw sequencer calculates frame buffer addresses, controls, color interpolation pacing and communicates with the memory hypervisor. The sequencer works with data path 370 to pass address information from the sequencer and corresponding data from the data path in a coordinated manner.

The input address parameters are processed to produce pixel addresses and write enables, using either the Bresenham or DDA algorithm for lines. The bitblt inputs determine which direction in X and Y that pixel reading and writing must proceed such that pixels are not written over before being copied. The input and output parameters for the sequencer are:

<table>
<thead>
<tr>
<th>Line Inputs:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ystart</td>
<td>Y1</td>
</tr>
<tr>
<td>Yend</td>
<td>Y2</td>
</tr>
<tr>
<td>Xstart,quot</td>
<td>X1 divided by five</td>
</tr>
<tr>
<td>Xend,quot</td>
<td>X2 divided by five</td>
</tr>
<tr>
<td>Xstart,rem</td>
<td>Remainder from division</td>
</tr>
<tr>
<td>Xend,rem</td>
<td>Remainder from division</td>
</tr>
<tr>
<td>D, 11, 12</td>
<td>Bresenham Algorithm</td>
</tr>
<tr>
<td>Slope_GT_1</td>
<td>Parameters</td>
</tr>
<tr>
<td>Slope_Pps</td>
<td></td>
</tr>
<tr>
<td>XI_GT_X2</td>
<td></td>
</tr>
<tr>
<td>Bibbit Inputs:</td>
<td></td>
</tr>
<tr>
<td>Xsrc_start,d5</td>
<td>Starting X address for the source block</td>
</tr>
<tr>
<td>Xsrc_end,d5</td>
<td>Ending X address for the source block</td>
</tr>
<tr>
<td>Ysrc_start</td>
<td>Starting Y address for the source block</td>
</tr>
<tr>
<td>Ysrc_end</td>
<td>Ending Y address for the source block</td>
</tr>
<tr>
<td>Xdest_start,d5</td>
<td>Starting X address for the destination block</td>
</tr>
<tr>
<td>Xdest_end,d5</td>
<td>Ending X address for the destination block</td>
</tr>
<tr>
<td>Ydest_start</td>
<td>Starting Y address for the destination block</td>
</tr>
<tr>
<td>Ydest_end</td>
<td>Ending Y address for the destination block</td>
</tr>
<tr>
<td>Xmasks (4)</td>
<td>5-bit Masks indicating which pixels in a 5-pixel group are to be manipulated at a block boundary.</td>
</tr>
</tbody>
</table>

Outputs:

| PixXaddr                | 8-bit X frame buffer address |
| PixYaddr                | 10-bit Y frame buffer address |
| WeMask                  | 5-bit pixel write enable mask |
| MemReadReq              | Frame buffer read request |
| MemWriteReq             | Frame buffer write request |
| DestNotSource           | Flag indicating if operation is destination or source. |

FIG. 8 is a more detailed block diagram of the input interface 300 shown in FIG. 2. An address decoder 700 includes a register 702 and a decoder 704. The decoder determines what type of request is pending and classifies the requests into different types such as overall request, triangle request, etc. There are also miscellaneous requests that are never protected. That is, an interrupt register can always be read or written. The request, read/write and strobe lines from the front end processor 210 and the busy signals from the rasterizer processor functional blocks are all inputs to a register protection checker 710. The register protection checker includes an interface 712 and an interface controller state machine 714. The checker compares the type of request with the current status of the busy lines to determine if the register operation can proceed. When the operation can proceed the ready line is asserted to the front end processor and (if the register operation is a write) the write strobe line is asserted to the correct functional block.

FIG. 9 is a state diagram of the input interface in operation. The command interpreter and supervisor 340, triangle interpolator 310, bitblt and line draw parameter calculator 320, and data path 370 each provide a busy signal to the input interface indicating when their internal registers are free to be read or written. When an internal register is written (or read) by the front end processor, the input interface determines if the operation may proceed by testing a busy signal from the associated functional block. If the register operation may proceed, then the input interface returns a ready signal to the front end processor. If the register operation is a read, then the data is sent to the front end processor. If the register operation is a write, then the internal register is strobed and the new data is loaded into it. If the register operation cannot proceed, then the input interface waits until the functional block permits the operation. The status of the busy signal of each block can be more than just a simple busy indicator. If the block has an input FIFO, then the busy can correspond to a FIFO full signal.

The register protection scheme improves overall performance by eliminating the testing of a status register. Whenever the next register load can proceed, the time spent testing the status register and determining that the register can be loaded is wasted. The performance improvement is most dramatic when only a few parameters need to be loaded for an operation and the operation is quick (such as a short line). In these cases, the time taken to read the status register indicating not busy is large relative to the time to load the parameters.

The register protection scheme also facilitates overlapping operations. For example, while the bitblt and line draw parameter calculator 320 is completing a line, parameters
can be loaded into the triangle interpolator. This increases the performance when there is a mixture of operations to be performed. When starting an operation, the command interpreter and supervisor 340 has a separate busy signal, thereby preventing update of the start operation register until the previous operation is complete.

While some registers are specific to one type of operation, others are common to more than one. The input interface allows each internal register to be protected by one or more busy signals. This allows, for example, the XY address of a line to be protected solely by the line drawer busy signal while the destination buffer selection (frame buffer, Z buffer or attribute buffer) requires all the busy signals to be inactive to prevent changing the buffer selection in the middle of an operation. Providing this differentiation allows a simple DMA controller to load any register in the rasterization processor in any order. The input interface prevents a register from being loaded at an improper time. The DMA controller can be set to copy from a display list in memory to the rasterization processor. As the rasterization processor completes operations, the next set of parameters can be loaded. Different operations can be mixed, as well as changes to overall control registers (buffer selection for example).

By separating address and data busy signals, such as the busy from the bitblt and line draw parameter calculator 320 and the busy from the data path 370, fast characters (bitblt) can be processed with a simple DMA interface. In this case, the protection mechanism allows the address information (destination X, Y) of each successive character to be entered while the previous character bitblt is completed. Then the protection mechanism will feed the data path at the rate it can handle the new character image.

Memory Hypervisor

There are several ways to implement Z and attribute compare in hardware. One is to have a single memory controller for the frame, attribute and Z buffers. The rasterizer front end generates address, color and Z depth information, decides which buffer currently needs to be accessed, and sends a request to the memory controller to access the appropriate buffer. When the memory controller has finished, the rasterizer sends the next request to be processed. While Z and/or attribute compare is active, several memory requests must be sent by the rasterizer front end to write a single pixel to the destination.

The performance of this implementation is limited primarily for three reasons. The first is that all buffer accesses must be done sequentially because there is only one memory controller. The second reason is that the process of reading the Z and/or attribute buffers, sending the data to the rasterizer front end, doing compares, deciding which buffer to access next, and sending the appropriate request to the memory controller requires many pipeline stages. This series or loop of operations must be performed for every pixel. The time required to go through this loop is almost always too long to operate at maximum memory bandwidth. The third reason is that the rasterizer front end cannot start rasterizing (i.e., generating address, color and Z depth) the next pixel until it finishes doing the Z and/or attribute compares and writes the current pixel to the destination buffer. Conversely, the Z and/or attribute compares cannot be performed unless that pixel has been rasterized. Since the time required to rasterize a pixel and to access a buffer varies, there will be many instances where the memory banks will be waiting for a pixel to be rasterized or the rasterizer will be waiting for the memory accesses to complete. These wait states reduce overall performance.

FIG. 11 is a more detailed block diagram of the memory hypervisor shown in FIG. 2. Parallel independent memory controllers 800, 810 and 820 are used for Z, attribute buffers, respectively. This makes it possible to execute multiple memory requests simultaneously such as reading the attribute buffer while writing the frame buffer. Even if two memory requests cannot be executed simultaneously because the buffers share data busses, it is possible to overlap the memory controllers such that the shared busses are run at maximum speed.

A hypervisor 830 is used to initiate and control the compare operations and memory controllers. It is close (i.e., few pipeline stages) to the memory controllers, thereby allowing the compare results to be utilized quickly. This allows us to more efficiently utilize the data busses.

Rasterizer front end requests are placed into a pipeline or buffer 840 in front of the hypervisor. This allows smoothing of the variations in performance between the rasterizer front end and the memory banks, thereby allowing each to operate at maximum speed more often. This reduces the number of instances where the memory banks are waiting for a pixel to be rasterized or the rasterizer front end is waiting for the memory accesses to complete. It also allows the hypervisor to start execution of the next memory operation while executing the current operation.

Read and write requests are sent by the rasterizer front end to the hypervisor. When the hypervisor is ready to accept a request, it will send a grant to the rasterizer front end and latch all necessary data associated with the request. Upon receiving a grant, the rasterizer front end considers the request complete and begins generating the next request, even though the hypervisor has not completed the first request.

In addition to read and write operations, refresh operations must be done on both VRAMs and DRAMs to ensure memory data integrity. Also, serial register load operations must be done on the VRAMs in order to display the memory contents on a graphics display. When either or both of these operations need to be performed, an interrupt is sent to the hypervisor. The hypervisor will complete the front end read or write request, cease operation until the required refresh/serial load operations are completed, and then resume processing rasterizer requests.

The pipelining, synchronization and sequencing of the memory requests are better understood with a few examples. FIG. 12 shows how the apparatus of FIG. 2 would process a series of writes to the frame buffer with Z and attribute compare being performed. First the rasterizer front end (FIG. 2) puts a write request into the pipeline. The hypervisor immediately determines which compare options must be performed (in this example, both attribute and Z). On the next cycle, a read request is sent to the attribute memory controller. Since this controller is inactive, it immediately sends a grant to the hypervisor and latches the address. The hypervisor considers the operation done upon receiving the grant and, on the next cycle, sends a read request to the Z memory controller. In this example, the Z and frame buffer memory share a common data bus, therefore the Z memory controller must wait until the frame buffer memory controller is done with the data bus. This wait period is determined by passing various busy signals between all the memory controllers. After the appropriate wait period, the Z memory controller sends a grant to the hypervisor. Next, a write
request is sent to the Z memory controller. If the Z and/or attribute compare is false, then the Z memory controller will abort the write operation after sending the grant. Finally, a write request is sent to the frame buffer controller. Again, if the compare is false, the frame buffer controller will abort the write operation after sending the grant.

FIG. 13 shows how the rasterizer would process a series of writes to the frame buffer with only an attribute compare being performed. First, the rasterizer front (FIG. 2) end puts a write request into the pipeline. The hyphervisor immediately determines which compare options must be performed (in this example, only attribute compare). On the next cycle, a read request is sent to the attribute memory controller. Since this controller is inactive, it immediately sends a grant to the hyphervisor and latches the address. Upon receiving this grant, the hyphervisor simultaneously sends a write request to the frame buffer memory controller and a read request to the attribute frame buffer (this attribute read sequence is as follows: the parameters of the triangle are one pipeline stage before the regular processor request). If the attribute compare is false, the frame buffer controller will abort the write operation after sending the grant.

Timing Diagrams

The operation of the pipeline and interlocks are better understood with a few examples. FIG. 14 shows how the rasterizer 220 of FIG. 2 would process a series of lines followed by a series of bitblts. The boxes denote when each subsystem is busy. The shading is to aid in differentiating between adjacent commands. For a line the sequence is as follows. The end points of the line are loaded through the input interface 300, the final value is a go signal to the command interpreter 340, the command interpreter starts the parameter calculator 320 if it is not either busy or waiting for its last set of calculated parameters to be used, the parameter calculator creates necessary values for the draw sequencer 360, and the draw sequencer makes memory requests to the memory hyphervisor 380. As soon as one subsystem completes the first line, it is free to work on the next line. Thus, the parameter calculator works on line 2 while the drawing sequencer is working on line 1. Once the first line is started, the memory hyphervisor is kept busy. If line 3 is followed by a bitblt, the command interpreter starts the parameter calculator on the bitblt even before the drawing sequencer is finished with a line. Therefore, two different styles of commands are processed in an overlapped fashion and the memory hyphervisor maintains maximum utilization.

FIG. 15 shows how the apparatus of FIG. 2 would process a series of triangles. The shading is to aid in differentiating between operations related to adjacent triangles. The sequencer are loaded through the input interface, the final value is the go signal to the command interpreter, the command interpreter starts triangle interpolator 220 if it is not busy, the triangle interpolator feeds a series of lines (or spans) to the parameter calculator if it is free, the parameter calculator creates the necessary values for the draw sequencer and the draw sequencer makes memory requests to the memory hyphvisor. There are two triangle parameter buffers, so as soon as one triangle is loaded, a second can also be loaded. The third triangle cannot be loaded until the first is completely interpolated. Once the first span of a triangle is started, the memory hyphvisor is kept busy. The triangle interpolator has internal storage for two sets of parameters so that there is not an unused stretch of time after one triangle is done while the input interface is busy receiving new parameters for the next triangle.

FIG. 16 shows how the apparatus of FIG. 2 would process a series of external bitblts which require format conversion of the input data (such as a character expansion). The shading is to aid in differentiating between operations related to adjacent bitblts. The sequence is as follows. The parameters of the bitblt are loaded through the input interface, the final value is the go signal to the command interpreter, the command interpreter starts the parameter calculator if it is free, the parameter calculator creates the necessary values for the draw sequencer and the format converter 330, the bitblt data can now begin to come via the input interface which passes it to the format converter, and the draw sequencer makes memory requests to the memory hyphvisor once the data path 370 has the data from the format converter. The format converter has an internal FIFO so that it can get ahead of the draw sequencer, thereby allowing the input interface to finish early. Then the parameters for bitblt 2 can be loaded restarting the whole sequence. If the format converter FIFO is large enough and the input interface loads faster than the memory hyphvisor can write the pixels, then the memory can be kept utilized. Certainly for characters this is true since for each bit of input data a pixel of output data is generated which is typically 3 bytes (24-bit pixels).

Overlapping operations are supported by having interlocks among the various state machines which allow subsequent operations to begin before the current operation is complete whenever possible. The input interface allows new values to be loaded in registers that are either currently not used or no longer required to be stable. The command interpreter determines which subsystems are involved in the previous command and allows a new command to proceed if the first active block for the new command is not busy processing the previous command. For example, as shown in FIG. 17, if the new command is a triangle and the previous command was a line (a bitblt would similar), then the command interpreter would pass the triangle interpolator the start triangle signal. The triangle interpolator would then proceed to create the first horizontal span of the triangle, but parameter calculation for that span would be held off until the bitblt and line draw parameter calculator was free. If the line is long enough (or the bitblt large enough), then the input interface loads the triangle before the memory hyphvisor completes the previous operation. The first line of the triangle can also be calculated while the previous line is being drawn, keeping the memory utilized. However, when a bitblt follows a triangle, the input interface prevents the bitblt parameters from being loaded until after the last line of the triangle is completed. This would leave a gap in the memory utilization depending on the length of the last line. In FIG. 17, a small gap is shown, but most lines are long enough so that no gap would exist.

Although the present invention has been fully described above with reference to specific embodiments, other alternative embodiments will be apparent to those of ordinary skill in the art. For example, the Bresenham algorithm for line draws can be broken into more than two sequential tasks that can be performed by more than two subsystems. Therefore, the above description should not be taken as limiting the scope of the present invention which is defined by the appended claims.

What is claimed is:
1. A data processing apparatus comprising:
   a plurality of processing means;
   storage means for receiving and serially storing a plurality of instructions from said plurality of processing means;
   a plurality of buffer memories, each buffer memory storing different types of data:
a plurality of buffer control means, each of said buffer control means coupled to a corresponding one of said buffer memories controlling access to that corresponding buffer memory for any of said plurality of processing means;

memory management means, connected to each of said buffer control means and to said storage means, for accessing a first instruction from a first one of said processing means and a second one of said plurality of instructions from said storage means from a second one of said processing means, and for obtaining from one or more of said buffer control means access to at least one first corresponding buffer memory in response to said first accessed instruction from said first processing means and updating at least one of said first corresponding buffer memory dependent upon a value previously stored in at least one of the accessed first corresponding buffer memory while, in response to said second instruction from said second processing means, concurrently obtaining from at least one buffer control means not currently accessed, access to at least one second corresponding buffer memory not currently accessed and updating at least one of said second corresponding buffer memories dependent upon a value previously stored in at least one of the accessed second corresponding buffer memories.

2. The data processing apparatus according to claim 1 wherein each of said buffer control means controls access to the corresponding buffer memory independent of other buffer control means.

3. The data processing apparatus according to claim 2 wherein said buffer control means are interconnected with each other to coordinate access to a data bus.

4. The data processing apparatus according to claim 3 wherein said storage means includes a first in and first out buffer.

5. The data processing apparatus of claim 1 wherein each said buffer memory includes storing different types of data corresponding to common pixel locations for concurrent access by said first and second processors.

6. The data processing apparatus of claim 5 wherein the first corresponding buffer memory includes storing a color type of data and the second corresponding buffer memory includes storing a depth type of data.

7. A method for controlling access of information in a memory apparatus comprising the steps of:

receiving from a plurality of processing means and serially storing a plurality of instructions in a storage means;

regulating access for any of said plurality of processing means to each of a plurality of buffers, each buffer storing different types of data;

accessing a first instruction from a first one of said processing means and a second instruction from a second processing means from the storage means, and obtaining access to one or more of said buffers in response to said first accessed instruction from said first one of said processing means and updating at least one of said first corresponding buffer memory dependent upon a value previously stored in at least one of the accessed first corresponding buffer memory while, in response to said second instruction, concurrently obtaining access to at least one buffer not already accessed and updating at least one of said second corresponding buffer memory dependent upon a value previously stored in at least one of the accessed second corresponding buffer memory.

8. The method of claim 7 wherein the step of regulating access to said plurality of buffers is performed by a plurality of buffer control means, each of said buffer control means regulating access for one of said plurality of buffers.

9. The method of claim 8 wherein each of said buffer control means regulates access to their respective buffer independent of the operations of remaining buffer control means.

10. The method of claim 8 wherein said step of storing the plurality of instructions in the storage means is performed using a first in and first out process.

11. The method of claim 7 wherein the step of regulating access includes storing different types of data corresponding to common pixel locations for concurrent access by said first and second processors.

12. The method of claim 11 wherein the step of regulating access includes storing a color type of data in the first corresponding buffer memory and storing a depth type of data in the second corresponding buffer memory.

13. A data processing system comprising:

a main processor;

a plurality of processing means for processing data under instruction of said main processor;

storage means for receiving and serially storing a plurality of instructions corresponding to buffer memory of processing means;

a plurality of buffer memories, each buffer memory storing different types of data;

a plurality of buffer control means, each of said buffer control means coupled to a corresponding one of said buffer memories for regulating access to that corresponding buffer memory for any of said plurality of processing means;

memory management means, connected to each of said buffer control means and to said storage means, for accessing through said storage means a first instruction from a first one of said processing means and a second instruction from a second one of said processing means, and for obtaining from one or more of said buffer control means access to at least one of first corresponding buffer memory in response to said first instruction from said first processing means and updating at least one of said first corresponding buffer memories dependent upon a value previously stored in at least one of the accessed first corresponding buffer memories.

14. The data processing system of claim 13 wherein each of said buffer control means controls access to the corresponding buffer memory independent of other buffer control means.

15. The data processing system of claim 14 wherein said buffer control means are interconnected with each other to coordinate access to a data bus.

16. The data processing system of claim 15 wherein said storage means includes a first in and first out buffer.

17. The data processing system of claim 13 wherein each said buffer memory includes storing different types of data corresponding to common pixel locations for concurrent access by said first and second processors.

18. The data processing system of claim 17 wherein the first corresponding buffer memory includes storing a color type of data and the second corresponding buffer memory includes storing a depth type of data.